Towards Prioritized Event Matching in a Content-based Publish/Subscribe System

Shiyou Qian, Jian Cao, Frédéric Le Mouël, Minglu Li, Jie Wang

Towards Prioritized Event Matching in a Content-based Publish/Subscribe System

Shiyou Qian
Department of Computer Science and Engineering, Shanghai Jiao Tong University
qshiyou@sjtu.edu.cn

Jian Cao *
Department of Computer Science and Engineering, Shanghai Jiao Tong University
cao-jian@sjtu.edu.cn

Frédéric Le Mouël
University of Lyon, INSA-Lyon, CITI-INRIA Lab
frederic.le-mouel@insa-lyon.fr

Minglu Li
Department of Computer Science and Engineering, Shanghai Jiao Tong University
mlli@sjtu.edu.cn

Jie Wang
Department of Civil & Environmental Engineering, Stanford University
jiewang@stanford.edu

ABSTRACT

QoS support is important for a large-scale content-based publish/subscribe (pub/sub) system to provide guaranteed service for clients with high QoS requirements. So far, great efforts have been dedicated to integrating QoS support into pub/sub systems. However, most work focus on providing QoS support on routing, without touching QoS support in event matching. In this paper, we propose the idea of prioritized event matching, aiming to integrate QoS support into event matching. We first point out the lack of time metrics that reveal performance detail of matching algorithms, leading to the definition of new time metrics. Through a series of experiments conducted in terms of new metrics, we discover the foundation for prioritized event matching. Finally, we realize prioritized event matching, called Pri-Rein, based on an existing matching algorithm and provide three design guidelines learned from the lessons in Pri-Rein. Extensive experiments are conducted to verify the effectiveness and efficiency of Pri-Rein and results show that Pri-Rein well achieves our design goal. We argue that the idea proposed in this paper can be generalized to matching algorithms that are used in cloud computing or complex event processing.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer Communication Networks - distributed systems

General Terms

Experimentation, Measurement, Performance

Keywords

Publish/Subscribe System, QoS, Event Matching, Priority

1. INTRODUCTION

How to quickly distribute data from providers to consumers is very challenging in a large-scale distributed system. In the extreme case where data are needed by all consumers, broadcasting is the simple and efficient way to disseminate data. On the other hand, if data are only interested by a few consumers, source routing is competent. The publish/subscribe (pub/sub) system is applicable to the situation between these two extreme cases where both broadcasting and source routing are not efficient. More importantly, the pub/sub system realizes the decoupling of communicating parties in time, space and synchronization [9]. The content-based pub/sub system is a flexible many-to-many communication model that meets the requirements of large-scale distributed applications, such as information filtering, selective content dissemination and location-based services.

A content-based pub/sub system consists of subscribers (data consumers), publishers (data providers), and an overlay network of brokers. The publishers input events into the system while the subscribers issue their subscriptions to the system. The core of a content-based pub/sub system is to propagate events from the publishers to the interested subscribers as quickly as possible. For a large-scale content-based pub/sub system, there may be millions of clients, including both subscribers and publishers.

With the increase of system scale, it is challenging to satisfy the QoS requirements of all subscribers in a large-scale content-based pub/sub system. Actually, overload is almost inevitable in large-scale distributed systems. For pub/sub systems, overload usually means the loss or delayed delivery of events. A common solution is to provide guaranteed service for subscribers with high QoS requirements. Therefore, providing QoS support is critical for content-based pub/sub systems to realize guaranteed delivery of

*Corresponding author.

DEBS'15, June 29 - July 3, 2015, OSLO, Norway.

ACM 978-1-4503-3286-6/15/06 ...$15.00.
http://dx.doi.org/10.1145/2675743.2771823.

events, attracting researchers great attention [2, 4, 8, 11, 18, 25, 28].

However, most existing work focus on providing QoS support on routing [4, 18, 25, 28], without touching QoS support in event matching. In this paper, we argue that subscribers with high QoS requirements expect to receive events earlier. The subscriptions issued by these subscribers are given high priority. To the best of our knowledge, there is no matching algorithm that takes the priority of subscriptions into account in event matching. It is also well recognized that event matching is a fundamental component in a content-based pub/sub system. A path for delivering events is composed of consecutive routes that are bridged by intermediate brokers at which event matching is performed. Due to the lack of QoS support in event matching, current solutions to QoS support in content-based pub/sub systems are partial and incomplete.

QoS support in event matching aims to earlier determine matched subscriptions with high priority than the ones with low priority. For example, consider a pub/sub system used in an emergency management application that provides information (events) to police and fire departments as well as the general public. Under normal conditions, the system can satisfy the QoS requirements of all subscribers. However, when an emergency happens, more people are concerned about the situation, resulting in large number of subscriptions which exceed the capacity of the system. With QoS support in event matching, events can be forwarded earlier to police and fire departments in the event matching process.

In this paper, we propose the idea of prioritized event matching, aiming to integrate QoS support into event matching. For prioritized event matching, subscriptions are differentiated according to their priorities, ensuring that matched subscriptions with higher priority are earlier determined than the ones with lower priority. Combined with QoS support on routing, prioritized event matching is capable of providing integral QoS support in content-based pub/sub systems, breaking through the point barriers in the delivery path of events.

We first point out the lack of time metrics that reveal performance detail of matching algorithms, highlighting the necessity to define new time metrics, including preparing time, ending time, gap time and interval time. These new metrics are much finer than traditional ones, well revealing the performance detail of matching algorithms. We conduct a series of experiments to evaluate five matching algorithms in terms of these new metrics, discovering the foundation for prioritized event matching. Based on an existing matching algorithm, Rein [20], we revise it to incorporate the priorities of subscriptions in the course of event matching. The new prioritized event matching algorithm is referred to as Pri-Rein.

Extensive experiments are conducted to evaluate the effectiveness and efficiency of Pri-Rein. Experimental results show that the idea proposed in this paper, namely prioritized event matching, is practicable for providing QoS support in event matching, well achieving our design goal. Compared with traditional matching algorithms, the determining time of matched subscriptions with high priority is significantly shortened in Pri-Rein. Our main contributions in this paper are:

- We point out the lack of time metrics that reveal the performance detail of matching algorithms. New time metrics are defined, including preparing time, ending time, gap time and interval time, for comprehensive performance evaluation of matching algorithms.
- Through a series of experiments, we discover the large time gap among the determining time of matched subscriptions. This discovery lays the foundation for prioritized event matching, aiming to integrate QoS support into event matching.
- Based on an existing matching algorithm, Rein, we realize the idea proposed in the paper, a prioritized event matching algorithm called Pri-Rein. Extensive experiments are conducted to verify the effectiveness and efficiency of Pri-Rein.

The rest of the paper is organized as follows. Section 2 defines some basic terms and the research problem of prioritized event matching. Section 3 discusses related work. Section 4 gives the foundation for prioritized event matching. The realization of prioritized event matching is described in Section 5. Section 6 presents the experimental results. The paper is concluded in Section 7.

2. PROBLEM FORMULATION

In this section, we first give the definition of terms used in content-based pub/sub systems. Then we formulate the problem of prioritized event matching. It is noted that lowercase t is used to represent specific moment in time and capital T to denote time (duration) in the paper.

2.1 Term Definitions

DEFINITION 1. *Events: An event is an observable occurrence, which is also called message or publication in some literatures. Clients who publish events are called publishers. Usually, an event is expressed as a conjunction of attribute-value pairs. Each attribute appears only once in an event expression. For example,* $\{(tem = 35), (hum = 15)\}$ *is an event describing the weather conditions. The set of attributes contained in the event expression is defined as* $A = \{a_1, a_2, ..., a_m\}$ *and the number of attributes in the set* A *is denoted by* m.

DEFINITION 2. *Primitive Constraints: A primitive constraint is a condition specified on an attribute in* A. *We consider range constraints in inclusive form in the paper, which are represented as a 4-tuple of* $\{att, value1, value2, type\}$. *Att is one of the attributes in* A. $Value1$ *and* $value2$ *are bounded by the value domain of the attribute and* $value1$ *is not larger than* $value2$. *Type defines the data type of the attribute with a value domain, which can be integer or double.*

DEFINITION 3. *Subscriptions: Clients who issue subscriptions are called subscribers. A subscription is an expression of subscribers' interests in events, which is also used to route and forward events from the publishers to the target subscribers. Each subscription has a unique subID and is specified as a conjunction of multiple primitive constraints. A subscription matches an event if all the primitive constraints contained in the subscription are satisfied when they are assigned the corresponding attribute values of the event.*

DEFINITION 4. *Event Matching: Given a set of n subscriptions $S = \{s_1, s_2, ..., s_n\}$ and an event e, event matching is to find all subscriptions from S that match e. The set of matched subscriptions S_m is a subset of S, $S_m \subseteq S$, $S_m = \{s_i \mid s_i \in S \cap s_i \text{ matches } e\}$.*

DEFINITION 5. *Matching Time: Given an event e and a set of n subscriptions $S = \{s_1, s_2, ..., s_n\}$, matching time is defined as the running time needed by a matching algorithm to match e against S. For example, if the matching algorithm starts and ends at moment t_s and t_e respectively, then the matching time T_M is defined as*

$$T_M = t_e - t_s. \quad (1)$$

DEFINITION 6. *Brokers: Brokers are also called servers, routers or proxies. In non-P2P computing environments, a broker is a specialized server that is responsible for routing and forwarding subscriptions and events. In P2P environments, some clients act as brokers.*

DEFINITION 7. *Selectivity: The selectivity of a constraint is the probability that the constraint is satisfied by assigning the corresponding attribute value in an event. The selectivity of a subscription is the probability that the subscription matches an event. The selectivity of a subscription is collectively determined by the selectivity of constraints contained in the subscription.*

2.2 Prioritized Event Matching

Based on the definition of event matching, we formulate the problem of prioritized event matching.

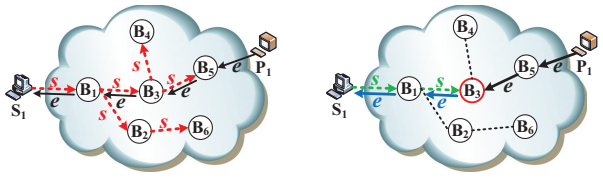
DEFINITION 8. *Prioritized Event Matching: Given an event e and a set of n subscriptions $S = \{s_1, s_2, ..., s_n\}$, each subscription s_i with a priority $p_i \in [P_l, P_h]$, where P_l and P_h are the lowest and highest level of priority, respectively. When matching e against S, each matched subscription is associated with a determining moment which represents the processing delay taken by the matching algorithm to find it out. Suppose there are k matched subscriptions $k \leqslant n$ and their determining moments are denoted by $\{t_f = t_1, t_2, ..., t_k = t_l\}$, where t_f and t_l denote the determining moment of the first and the last matched subscription, respectively. Let s_{t_i} be the matched subscription that is found at moment t_i for $1 \leqslant i \leqslant k$, $s_{t_i} \in S$. Prioritized event matching guarantees that if s_{t_i} has higher priority than s_{t_j}, t_i must be smaller than t_j. In other words, matched subscriptions with high priority must be picked out earlier than the ones with low priority.*

3. RELATED WORK

In this section, we briefly review routing strategies employed in content-based pub/sub systems, emphasizing the fundamental role of event matching. Then we review existing research on matching algorithms and QoS support in content-based pub/sub systems.

3.1 Routing Strategies

Large-scale content-based pub/sub systems usually disseminate events through an overlay network of brokers. To construct a delivery path, subscriptions and events must meet at least one broker to perform event matching and transmission. According to the way to construct delivery paths, existing routing strategies can be roughly classified into two categories: broadcast routing and rendezvous routing.



(a) Broadcast routing (b) Rendezvous routing

Figure 1: Routing strategies used in content-based pub/sub systems

Elvin [24], JEDI [7] and Siena [5] are representative pub/sub systems that broadcast subscriptions (or advertisements) in the overlay network of brokers. Events are delivered on the reverse paths constructed in the broadcasting process. Along a delivery path, event matching is performed at each broker to determine whether to forward events or not. Figure 1 (a) depicts an example of broadcast routing. In the example, the overlay network is composed of six brokers, from B_1 to B_6. The subscriber S_1 issues a subscription s which is broadcast in the overlay network (denoted by the red dashed arrow lines). The publisher P_1 publishes an event e that matches the subscription s. The event e is delivered to S_1 on the reverse path constructed in the broadcasting process (denoted by the black arrow lines). In this case, event matching is performed three times at broker B_5, B_3 and B_1, resulting in high end-to-end latency.

Rendezvous routing was first proposed in topic-based pub/ sub systems, such as Scribe [22] and Hermes [19]. Meghdoot [12] adapted this routing strategy to content-based pub/sub systems, aimed at reducing the times of event matching on the delivery paths. For rendezvous routing, a subscription is directly sent to a rendezvous broker that is determined by using distributed hash table (DHT). Similarly, an event is also sent to a rendezvous broker at which event matching is performed to decide whether to forward the event or not. Figure 1 (b) shows an example of rendezvous routing. The subscriber S_1 directly submits his subscription s to the rendezvous broker B_3 (denoted by the green dashed arrow lines) and the publisher P_1 publishes the event e to broker B_3 (denoted by the black arrow lines). Broker B_3 performs event matching and delivers e to S_1 (denoted by the blue arrow lines). Through rendezvous routing, the times of event matching is reduced from three in broadcast routing to one.

We will not discuss the pros and cons of the two routing strategies. We just emphasize that event matching is essential to a content-based pub/sub system, no matter which routing strategy is adopted. In fact, since the emergence of content-based pub/sub systems, event matching is one of the main research topics.

3.2 Matching Algorithms

Content-base pub/sub systems are more flexible than topic-based ones. However, it is also recognized that the cost of event matching in content-based pub/sub systems is pretty higher than that in topic-based systems. Over the past two decades, much effort has been made to design

efficient matching algorithms [1, 3, 6, 10, 14–17, 20, 21, 23, 26, 27]. In these algorithms, different data structures are used to index subscriptions for the purpose of speeding up event matching. The underlying data structures include matching tree [1], matching table [6, 27], binary decision diagram [3, 17], BE-Tree [23], OpIndex [26] and bloom filter [16].

A common feature of these matching algorithms is that subscriptions are decomposed into primitive constraints which are indexed in the underlying data structures. Since primitive constraints contained in a subscription are processed separately in the data structures, it is very challenging to provide QoS support in event matching. As far as we know, there is no matching algorithm that takes QoS support into consideration in event matching.

3.3 Quality of Service

Providing QoS support in pub/sub systems has been widely studied [2, 4, 8, 11, 13, 18, 25, 28]. The Java Message Service (JMS) provides a way to deliver expedited messages ahead of normal messages, defining a ten level priority value with 0 as the lowest priority and 9 as the highest [13]. IndiQoS, presented in [4], is a distributed and scalable pub/sub broker with support for QoS, which automatically selects QoS-capable paths by leveraging existing mechanisms to reserve resources in the underlying network and on an overlay network of P2P rendezvous nodes. The scalability of IndiQoS is guaranteed by avoiding the flooding of either QoS reservations or link-state information. In [28], constraint-based routing was proposed to take into account the subscribers' QoS requirements, the dynamic characteristics of overlay network (such as latency and bandwidth) and events quality in the process of routing, besides the subscription information. The objective is to determine a routing path connecting a publisher and a subscriber, fully satisfying the QoS requirements.

To deal with the unreliable delivery in content-based pub/ sub systems, Malekpour et al. design an end-to-end, probabilistic reliable service to enhance the reliability of best-effort content-based pub/sub systems in [18]. The purpose is to make a balance between high throughput and end-to-end delivery delay. In [11], a novel methodology is presented for the configuration of pub/sub systems that support developers with selecting the QoS-optimal configuration, addressing the problem of choosing an optimal configuration for pub/sub systems.

However, there is little work focusing on providing QoS support in event matching. In fact, the delivery paths of events are composed of consecutive routes that are bridged by brokers at which event matching is performed to decide whether to forward events or not. Without considering QoS support in event matching, brokers become point barriers of an integral QoS support solution. In the paper, we aim to break through these point barriers by integrating QoS support into event matching.

4. INSIGHT INTO EVENT MATCHING

In this section, we first point out the fact that existing time metrics fail to provide performance detail of matching algorithms. Complying with the evaluation criteria, we define new time metrics that reveal more detail on the performance of matching algorithms. Through experiments, we discover the foundation for prioritized event matching.

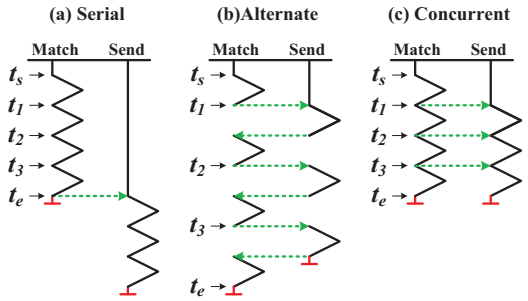


Figure 2: Three running ways of match and send procedures

4.1 Lack of Fine Grained Time Metrics

To disseminate an event to all interested subscribers, a broker first needs to perform event matching to get the matched subscriptions for the event, and then sends the event to the subscribers that issue matched subscriptions. Match and send procedures can run in three ways, namely serial, alternate or concurrent, as shown in Figure 2. In the figure, there are three matched subscriptions for an event, determined at moment t_1, t_2 and t_3, respectively. For a matched subscription, determining time (or delay) is referred to the time needed to pick it out as matched by the matching algorithm; processing time (or delay) is defined as the total time staying at the broker, which contains the determining time.

Serial way has short determining time for matched subscriptions, but at the cost of long processing time. To reduce processing time, alternate way trades determining time for the improvement on processing time. With the help of parallel computing power of hardware, concurrent way conquers the drawback of serial way, improving processing time without sacrificing determining time. However, it is obvious that matching time fails to reveal the determining time of matched subscriptions, no matter which running way is applied. In fact, matching time is a coarse metric to evaluate the performance of matching algorithms, mostly concerned with event matching throughput which is defined as the number of events matched in a unit time. Generally, throughput is inversely proportional to matching time.

Given an event, suppose there are k matched subscriptions. The determining moments of the k matched subscriptions are denoted by $\{t_f = t_1, t_2, ..., t_k = t_l\}$, where t_f and t_l denote the determining moment of the first and the last matched subscription, respectively. The determining time of the i^{th} matched subscription T_{D_i} for $1 \leqslant i \leqslant k$ is defined as

$$T_{D_i} = t_i - t_s, \tag{2}$$

where t_s is the starting moment of event matching. Specifically, we use T_{D_f} and T_{D_l} to represent the determining time of the first and the last matched subscription, respectively. The definition of determining time for matched subscriptions is necessary to gain insight into the performance detail of matching algorithms.

4.2 Evaluation Criteria and New Metrics

Traditionally, matching algorithms are theoretically analyzed in terms of time and space complexities and experimentally evaluated in terms of matching time, insertion

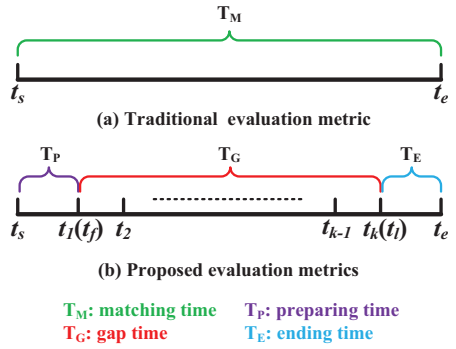


Figure 3: Traditional and proposed time metrics for performance evaluation of matching algorithms

time, deletion time and memory occupation. On the basis of traditional evaluation metrics, we propose three new evaluation criteria and corresponding time metrics.

4.2.1 Criteria 1: High Event Matching Throughput

When matching an event against a set of subscriptions, the number of matched subscriptions may be 0, 1, or > 1. No matter the number of matched subscriptions, it is ideal to finish event matching as quickly as possible in order to improve event matching throughput. For matching algorithms, throughput is defined as the number of events matched in a unit time. Traditional matching time T_M defined in Definition 5 is sufficient to evaluate the throughput of matching algorithms.

4.2.2 Criteria 2: Low Maintenance Cost

In some dynamic environments, subscriptions may be frequently updated. This dynamism requires matching algorithms to provide not only high event matching throughput but also low maintenance cost. Insertion time and deletion time are used to measure the maintenance cost of matching algorithms. Insertion time is used to measure the cost to insert a new subscription into the underlying data structures of matching algorithms and deletion time is the time taken to delete an existing subscription from the data structures. Since we mainly focus on the matching performance of algorithms, maintenance cost is not discussed in this paper.

4.2.3 Criteria 3: Short Preparing Time

As defined in Definition 5 and depicted in Figure 3(a), matching time is a coarse metric in nature which represents the time needed to match an event against a set of subscriptions. Matching time does not reveal details on the matching process. In the case where an event is not subscribed by any clients, matching time is sufficient to evaluate matching algorithms. However, in large-scale pub/sub systems, a more common case is that an event is subscribed by multiple clients. In fact, almost all matching algorithms are based on their specialized data structures to speed up event matching. Before determining the first matched subscription, some time is needed for matching algorithms to perform operations on the underlying data structures. This preparation time is referred to as preparing time T_P, which is defined as

$$T_P = t_f - t_s \tag{3}$$

as shown in Figure 3(b). Preparing time is the common matching delay for all matched subscriptions. Low preparing time means low matching delay to forward events to subscribers.

4.2.4 Criteria 4: Short Ending Time

Usually, matching algorithms do not immediately terminate after the last matched subscription is determined. Since matching algorithms do not know the exact number of matched subscriptions before matching, additional processing is necessary to prevent the occurrence of false negatives. Here, a false negative means the occurrence that a matched subscription is not found by the matching algorithm. We define ending time T_E as the time spent on additional processing after the last matched subscription is found.

$$T_E = t_e - t_l \tag{4}$$

For matching algorithms, shortening ending time is beneficial to improve matching throughput.

4.2.5 Criteria 5: Uniform Determining Interval

When matching an event against a set of n subscriptions, suppose there are k matched subscriptions and $k \leqslant n$. As mentioned above, matching time T_M is not sufficient to reveal the details on the determining time of matched subscriptions. For concurrent running way shown in Figure 2 (c), the intervals between the determining times of consecutive matched subscriptions should be small and uniform. Given k matched subscriptions, there are $k-1$ intervals. We define $I_{i,j}$ as the interval between the determining times of the i^{th} and j^{th} matched subscription,

$$I_{i,j} = T_{D_j} - T_{D_i}, \tag{5}$$

where $1 \leqslant i < k$, $1 < j \leqslant k$ *and* $j = i + 1$. We define interval time T_I as the average of the $k-1$ intervals.

$$T_I = \frac{\sum_{i=1}^{k-1} I_{i,i+1}}{k-1} \tag{6}$$

4.3 Foundation for Prioritized Event Matching

It is ideal to find out all matched subscriptions at the same moment when matching an event. However, even though subscriptions are divided into groups and parallel matching algorithms are employed to search matched subscriptions, there still exists sequential order in determining the matched subscriptions. Furthermore, the difference between the determining times of the first matched subscription and the last one may be large when the number of subscriptions and the selectivity of subscriptions are high in a content-based pub/sub system. We define a time metric, called gap time T_G, to measure this difference,

$$T_G = T_{D_l} - T_{D_f} \tag{7}$$

where T_{D_l} and T_{D_f} are the determining time of the last and the first matched subscription, respectively. T_G represents how much more time is taken to determine the last matched subscription than the first one.

However, it is the gap time that provides the foundation for QoS support in event matching. If all matched subscriptions are picked out at the same moment, it is not possible to differentiate subscribers with different QoS requirements

Table 1: Parameters used in the experiments

Name	Meaning
n	the number of subscriptions
m	the number of attributes in events
r	the number of constraints in subscriptions
w	the width of range constraints
c	the number of cells in H-Tree
l	the number of index attributes in H-Tree
d	the discretization level in Tama
b	the number of buckets in Rein

in event matching. We argue that subscriptions issued by subscribers that have high QoS requirements are usually given high priority. The basic idea of prioritized event matching is that the priority of subscriptions should be considered in the course of event matching. So, matched subscriptions with high priority should be picked out earlier than the ones with low priority.

In the following, we conduct a series of experiments to evaluate the matching performance of five algorithms in terms of the new time metrics defined, aiming at demonstrating the existence of large gap time.

4.3.1 Experimental Setup and Parameters

The tested matching algorithms include Simple, Siena, Tama, H-tree and Rein. Simple is used as the baseline, which utilizes a naive matching policy by comparing an event with subscriptions one by one. Siena is a two-level forwarding table, the first level indexed on attributes, and the second level indexed on operators [6]. Tama is an approximate matching and forwarding engine which uses a hierarchical matching table to store subscriptions [27]. Tama trades matching accuracy for improvement on matching time. H-Tree is an exact matching algorithm which indexes similar subscriptions in the same bucket of its underlying data structure [21]. H-tree filters most of unmatched subscriptions in the searching process, leaving a small proportion of subscriptions to compare using naive matching policy. Compared with the other four algorithms, Rein adopts a reverse searching strategy to find matched subscriptions [20]. In the matching process, Rein identifies and marks unmatched subscriptions in a bitset, with the unmarked bits representing matched subscriptions. Parameters used in the experiments are listed in Table 1.

All experiments are conducted on a Dell PowerEdge T710 running Ubuntu 12.04.5 LTS with Linux kernel 3.2.0-75-generic, which has 8 2.4GHz cores and 32GB memory. Parallelism is not used in all experiments. All codes are written in C++ language. In the experiments, the value domain of attributes is normalized on [0, 1.0] for brevity. The constraint values and event values are uniformly generated from this domain with precision of 10^{-6}. The constraint attributes are uniformly selected from event attributes. For Rein, real values are converted into integer with the availability of value domain and precision. The discretization level in Tama is set to $d = 13$. The number of cells and index attributes in H-tree is set to $c = l = 6$. The number of buckets in Rein is set to $b = 1000$. In each experiment, 500 events are matched and the average is computed for preparing time T_P, ending time T_E, matching time T_M, gap time T_G and interval time T_I.

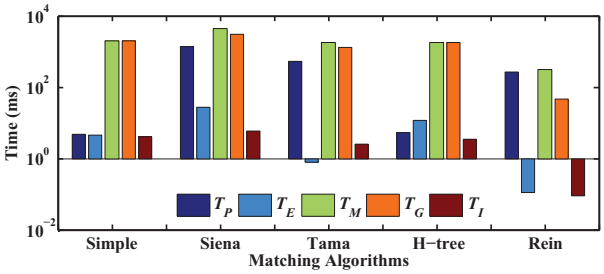


Figure 4: The preparing time(T_P), ending time(T_E), matching time(T_M), gap time(T_G) and interval time(T_I) of the tested matching algorithms

Table 2: The standard deviations of T_P, T_E, T_M, T_G and T_I of the tested matching algorithms

	T_P	T_E	T_M	T_G	T_I
Simple	5.55	5.84	80.99	81.81	4.37
Siena	286.34	25.20	73.27	303.17	26.92
Tama	98.10	0.82	111.56	147.01	8.14
H-tree	5.50	22.59	128.84	128.45	4.84
Rein	4.56	0.14	4.51	0.22	0.10

4.3.2 An Intuitive Image

In the first experiment, we verify the existence of gap time for the five tested matching algorithms. The parameters are set as follows: $n = 500000$, $m = 50$, $r = 5$ and $w = 0.25$. The average selectivity of subscriptions is about 0.001. On average, each event matches 500 subscriptions. The results of T_P, T_E, T_M, T_G and T_I are depicted in Figure 4 with the y-axis representing time in log-scale. Since the values of standard deviations are much smaller compared with the values of T_P, T_E, T_M, T_G and T_I, it is meaningless to draw standard deviations in the form of error bars. The standard deviations of T_P, T_E, T_M, T_G and T_I are listed in Table 2.

By observing the figure, we find that gap time does exist in matching algorithms, sometimes even approaching to matching time. The ratio between gap time and matching time is 99.5%, 68.6%, 71.2%, 99.1% and 14.9% for Simple, Siena, Tama, H-tree and Rein, respectively. For Simple, there is no preparing stage, just matching an event against subscriptions one by one. In the extreme case where the first and the last subscription match events, the gap time of Simple is equal to the matching time. For H-tree, its underlying data structure contains a certain number of buckets and the subIDs of similar subscriptions are stored in a bucket. The preparation of H-tree amounts to computing the buckets to be checked, which is very fast. After determining the buckets, naive matching strategy is used to search matched subscriptions, which is similar to Simple. So, Simple and H-tree behave similarly in terms of the ratio between gap time and matching time. Even through the ratio of Rein is the smallest among the five matching algorithms, its gap time still reaches 47 milliseconds (ms) on average.

The second observation made from Figure 4 is that matching algorithms that employ specialized data structures need long preparing time that is necessary to realize fast matching. For Simple, event matching is performed without a data structure. So, its preparing time is very small, less

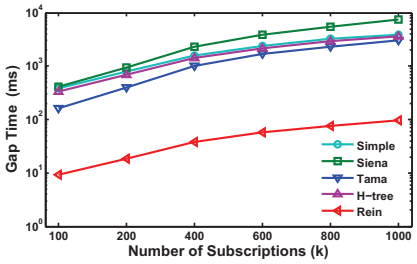


Figure 5: Gap time with the number of subscriptions

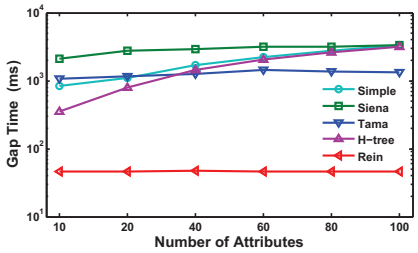


Figure 6: Gap time with the number of attributes

Table 3: The standard deviations of gap time (T_G) with the number of subscriptions

	100k	200k	400k	600k	800k	1000k
Simple	9.9	12.6	45.4	31.5	83.5	47.9
Siena	61.0	122.8	243.3	361.8	494.9	601.3
Tama	26.5	59.7	122.8	197.0	261.0	311.0
H-tree	25.3	48.9	108.9	159.4	200.9	249.1
Rein	0.2	0.2	3.0	4.3	5.7	6.3

than 5 ms. The underlying data structure of H-tree is used to store the subIDs of subscriptions rather than the content of subscriptions, resulting in small preparing time, which is similar to Simple. For Siena, Tama and Rein, the ratio between preparing time and matching time is 30.8%, 28.8% and 85.1%, respectively. The high ratio of Rein makes it be the fastest among the five tested matching algorithms, significantly improving matching efficiency.

The third observation is that ending time occupies a small proportion of matching time for the five tested algorithms. The proportion is 0.23%, 0.62%, 0.04%, 0.66% and 0.03% for Simple, Siena, Tama, H-tree and Rein, respectively. After finding the last matched subscription, Rein finishes event matching very quickly, just taking 0.11 ms. This merit is contributable to the efficient bit operations used in Rein.

Finally, we find that interval time is directly related to gap time. Longer gap time usually means larger interval time. Siena has the largest interval time, reaching 6.03 ms. Moreover, the interval time of Siena fluctuates seriously, resulting in high standard deviation as shown in Table 2. The standard deviation of the interval time of Rein is just 0.10, showing that the interval time of Rein is short and uniform.

Figure 4 gives an intuitive image of the matching performance of the five tested algorithms. Although other time metrics are important to comprehensively evaluate matching algorithms, we mainly focus on gap time since it provides the foundation for prioritized event matching. Gap time is affected by multiple factors, including the number of subscriptions, the number of attributes, the number of constraints contained in subscriptions and the selectivity of subscriptions. The effects of these factors are evaluated in the following subsections.

4.3.3 *The Number of Subscriptions*

Generally, gap time increases with the number of matched subscriptions. Given the selectivity of subscriptions, the number of matched subscriptions is proportional to the number of subscriptions. Therefore, gap time grows linearly in the number of subscriptions. Figure 5 confirms this

Table 4: The standard deviations of gap time (T_G) with the number of attributes

	10	20	40	60	80	100
Simple	35.7	20.5	63.3	26.4	67.6	68.1
Siena	385.0	346.2	298.6	319.9	311.9	346.7
Tama	260.8	198.8	160.4	186.5	148.8	133.7
H-tree	180.4	160.8	135.0	123.3	128.7	136.7
Rein	1.3	0.3	3.2	0.2	0.2	0.2

analysis, where the y-axis represents gap time in log-scale. In the experiment, the parameters are set as follows: $m = 50$, $r = 5$ and $w = 0.25$. On average, the gap time of Simple, H-tree and Rein grows almost 100% when the number of subscriptions is doubled. As for Siena and Tama, the growth rate is about 135%. When $n = 1000k$, the gap time of Rein is 95.40 ms, the least one among the five tested matching algorithms; while Siena has the largest gap time, reaching 7346.43 ms. The standard deviations of gap times are listed in Table 3. In general, the standard deviation increases with the gap time for the five tested matching algorithms.

4.3.4 *The Number of Attributes*

Whether the number of attributes affects gap time or not depends on the underlying data structure used by matching algorithms. The results are shown in Figure 6 with the y-axis representing gap time in log-scale. In the experiment, the parameters are set as follows: $n = 500000$, $r = 5$ and $w = 0.25$. Among the five tested matching algorithms, H-tree is most affected by the number of attributes. When the number of attributes is doubled, the gap time of H-tree increases 91.52% on average. On the contrary, Rein is almost unaffected. For Simple, Siena and Tama, the growth rate of gap time is 61.34%, 14.46% and 4.60% respectively, when the number of attributes is doubled. The standard deviations of gap times are listed in Table 4 for the five tested algorithms.

In addition, the number of constraints contained in subscriptions has an impact on gap time, which is very similar to the one shown in Figure 6. Due to space limitation, the results are omitted in the paper.

4.3.5 *The Selectivity of Subscriptions*

Given the number of subscriptions, high selectivity amounts to more matched subscriptions, which in turn lead to the growth of gap time. The results of testing the effect of the selectivity of subscriptions are shown in Figure 7 with the y-axis representing gap time in log-scale. In the experiment, the parameters are set as follows: $n = 500000$, $m = 50$, $r = 5$. The width of range constraints changes with the selectivity of subscriptions. Among the five tested

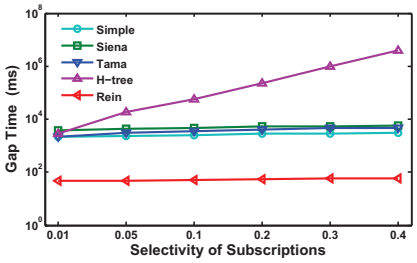


Figure 7: Gap time with the selectivity of subscriptions

Table 5: The standard deviations of gap time (T_G) with the selectivity of subscriptions

Selectivity	Simple	Siena	Tama	H-tree	Rein
0.01	60.2	244.5	225.3	724.1	0.1
0.05	139.1	240.9	297.0	13690.2	0.6
0.1	164.8	223.5	311.0	44016.3	1.3
0.2	182.8	235.3	314.0	157162.2	2.3
0.3	185.0	227.2	300.1	322071.3	2.9
0.4	201.8	215.0	277.6	745730.8	3.3

matching algorithms, the gap time of H-tree is most affected, increasing exponentially with the selectivity of subscriptions. The gap time of Simple, Siena and Tama grows linearly with the selectivity of subscriptions. Rein performs best with small variation, compared to other four algorithms. For example, when the selectivity is 0.01 and 0.4, the gap time of Rein is 47.04 ms and 59.99 ms, respectively. The standard deviations of gap times are listed in Table 5.

5. PRIORITIZED EVENT MATCHING

As demonstrated by the experimental results in Section 4.3, Rein has shorter preparing time, ending time and matching time, compared with other four algorithms. Meanwhile, the interval time of Rein is short and uniform, which is suitable for concurrent running of match and send procedures. We realize prioritized event matching by embedding QoS support into Rein.

Concerning the delivery of events in a content-based pub/sub system, delivery delay consists of processing delay, queuing delay and transfer delay. Queuing delay and transfer delay are related to scheduling process and routing process respectively, which are out of the scope of this paper. Processing delay is mainly caused by event matching and sending at brokers. Prioritized event matching ensures that matched subscriptions with high priority should be picked out earlier than the ones with low priority, aiming to reduce the determining time of matched subscriptions with high priority.

5.1 Preliminary knowledge of Rein

No matter the underlying data structures used, most matching algorithms adopt forward searching strategy, ignoring unmatched subscriptions in the course of event matching. On the contrary, Rein applies reverse searching strategy to find matched subscriptions for events. The matching course of Rein can be divided into two stages, preparing stage and output stage. In the preparing stage,

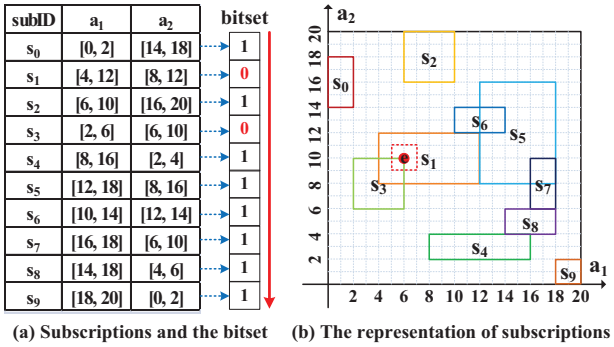


Figure 8: The basic idea of Rein

unmatched subscriptions are searched and marked in a bitset. In the output stage, the bitset is checked and the unmarked bits in the bitset represent matched subscriptions, as shown in Figure 8 (a). One advantage of reverse searching is that, given the number of subscriptions, the matching time of Rein decreases with the number of matched subscriptions. In other words, the selectivity of subscriptions does not affect the matching performance of Rein. As shown in Figure 7, the matching performance of Simple, Siena, Tama and H-tree degrades when the selectivity of subscriptions increases.

Rein constructs its underlying data structure based on range constraints. Since subscriptions are composed of range constraints, the event matching problem is equivalent to the point enclosure problem. The attributes contained in events form a high-dimensional space in which a subscription is a high-dimensional rectangle and an event is a point. Rein transforms the point enclosure problem into the rectangle intersection problem by enlarging a point into a cube. During the transformation, constraint values and event values are doubled to prevent false positives. The purpose of the transformation is to utilize the quick determination method of two disjoint rectangles. For example, given the two rectangles s_1 and s_2 depicted in Figure 8 (b), the bottom side of s_2 lies on the top of s_1's top side, so it is very fast to judge that s_1 disjoints s_2 by efficient comparison operations.

5.2 Integrating QoS Support in Rein

It is straightforward to realize prioritized event matching based on Rein. As mentioned above, the matching course of Rein is divided into two stages, preparing stage and output stage. During the preparing stage, Rein identifies and marks unmatched subscriptions in a bitset with the help of its underlying data structure that is constructed based on range constraints. We realize prioritized event matching in the output stage, without touching the underlying data structure of Rein. The basic idea of realizing prioritized event matching based on Rein is very simple, by changing the checking order of the bitset.

As shown in Figure 8, when outputting matched subscriptions, Rein scans the bitset from the first bit to the last one (denoted by the arrow line in Figure 8 (a)) without taking the priority of subscriptions into account. To support prioritized event matching, it is necessary to first output matched subscriptions with the highest priority. The realization of prioritized event matching based on Rein is shown in Figure 9, adding a list to store the checking order in the bitset for each level of priority (denoted by the

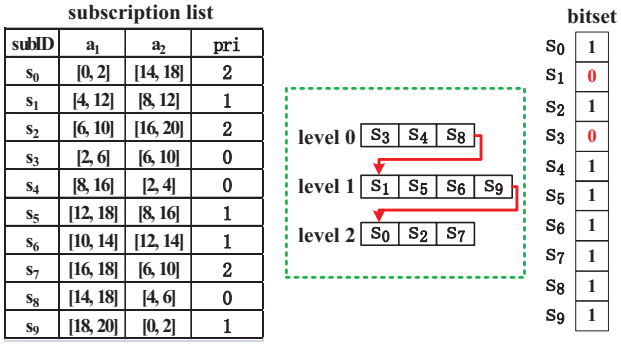


Figure 9: The basic idea of Pri-Rein

dashed rectangle). Subscriptions with the same priority are stored in a list. When outputting matched subscriptions, subscriptions with the highest priority are first checked in the bitset. For example in Figure 9, there are three levels of priority, from 0 to 2 as shown in the subscription list. The number of subscriptions corresponding to the three level of priority is 3, 4 and 3, respectively, as shown by the three lists in the middle. Given an event $e = \{a_1 = 6,\ a_2 = 10\}$, the subscriptions that match e are s_1 and s_3, and s_3 has higher priority than s_1. In the output stage of event matching, the subscriptions with the highest priority are first checked (denoted by the two arrow lines in the dashed rectangle), so s_3 is first picked out as matched and then is s_1. This new prioritized event matching algorithm is referred to as Pri-Rein. The pseudo code of Pri-Rein is shown in Figure 10.

Algorithm 1: Pri-Rein

```
1: input: an event e
2: output: matched subIDs ID
3: initialize a bitset of size n;
4: add subscriptions' subID to corresponding priority list;
5: mark all unmatched subscriptions in the bitset;
6: for (from the highest to lowest level of priority) do
7:    for (each subID in the priority list ) do
8:       if (the bit corresponding to the subID is unmarked)
         then
9:          add subID to ID;
10:      end if
11:   end for
12: end for
13: output ID;
```

Figure 10: The prioritized event matching algorithm

5.3 Design Guidelines

Learned from the lessons in Pri-Rein, we propose three design guidelines for realizing prioritized event matching.

5.3.1 *Associating Priority with Subscriptions*

The first guideline of integrating QoS support in event matching is that priority should be associated with subscriptions, rather than primitive constraints. As defined in Definition 3, a subscription is composed of multiple primitive constraints. Most matching algorithms construct their underlying data structures based on primitive constraints. For example, matching tree and matching table are used to index primitive constraints in [1] and [6], respectively. If priority is associated with primitive constraints, the priority of a subscription is stored multiple times, at least equal to the number of constraints contained in the subscription, sometimes even far larger. The consequence of repeatedly storing is the increase of space consumption, growing linearly in the number of primitive constraints.

Furthermore, if priority is associated with primitive constraints, priority conflict may appear on a primitive constraint that is contained in multiple subscriptions. A common case is that a primitive constraint has multiple different priorities since it is contained in multiple subscriptions that have different priority. Even worse in a content-based pub/sub system where the set of primitive constraints is fixed and the number of subscriptions is large, each primitive constraint is contained in at least one subscription with the highest priority. In this extreme case, it is impossible to realize QoS support in event matching by associating priority with primitive constraints.

5.3.2 *Combining Priority with Selectivity*

Combining priority with selectivity is the second guideline, aiming to reduce the determining time of matched subscriptions with high selectivity. Without considering the selectivity of subscriptions in prioritized event matching, a common way to order subscriptions with same priority is the FCFS (First-Come First-Served) policy. As a result, much more time may be wasted on unmatched subscriptions that have low selectivity but are in the front of the order. This case may meaninglessly increase the determining time of matched subscriptions with high selectivity, degrading the overall performance of matching algorithms. Therefore, selectivity and priority should be considered simultaneously in prioritized event matching to further optimize the matching performance.

Moreover, the guideline of combining priority with selectivity verifies the necessity of the first guideline, namely associating priority with subscriptions. When matching an event with a subscription, only all primitive constraints contained in the subscription are satisfied, we say the event matches the subscription. The selectivity of a subscription is collectively determined by the selectivity of primitive constraints contained in the subscription, not single ones. A high selective primitive constraint does not necessarily mean the high selectivity of the subscription containing it. Embedding priority into primitive constraints may result in wasting time on checking less selective subscriptions that contain a few high selective primitive constraints. In this case, the determining time of matched subscriptions is increased. Therefore, subscriptions and their selectivity should be the atomic units to be processed in prioritized event matching.

5.3.3 *Balancing between Gains and Losses*

It is obvious that providing QoS support in event matching is not costless. On one side, the gains are the better service for subscribers with high QoS requirements, improving users' experience and satisfaction. On the other side, the losses are the increase of matching time due to the additional processing of priority in event matching. However, increasing matching time means processing less

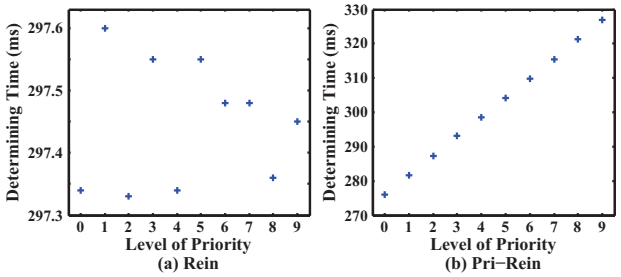


Figure 11: The correlation between the level of priority and determining time

events in a unit time, resulting in lower event matching throughput. If the losses overwhelm the gains, causing a substantial decline in event matching throughput, it is worth rethinking whether to provide QoS support or not in event matching, or considering a better solution. Therefore, last but not least, keeping a balance between gains and losses is the third guideline for prioritized event matching.

6. EXPERIMENTS

To the best of our knowledge, there are no matching algorithms that consider the priority of subscriptions in event matching. Thus, we demonstrate the effectiveness and efficiency of Pri-Rein by comparing it with Rein. The details of experimental setup and parameters are given in Section 4.3.1.

6.1 Pri-Rein without Optimization

In this experiment, subscriptions with same selectivity are tested. In each level of priority, subscriptions are ordered according to the FCFS policy.

6.1.1 *Effectiveness*

To demonstrate the effectiveness of Pri-Rein, we record the determining time and priority of each matched subscription. The level of priorities is set to 10 in the experiment, from the highest 0 to the lowest 9. The priority of each subscription is uniformly generated from [0, 9]. Other parameters are set as follows: $n = 500000$, $m = 50$, $r = 5$ and $w = 0.25$. All subscriptions have the same selectivity of 0.001. 5000 events are matched against the set of subscriptions.

Overall, the average determining time of all matched subscriptions is 297.45 ms and 301.47 ms for Rein and Pri-Rein, respectively, with only 1.35% difference. The standard deviation of determining times is 18.12 and 19.56 for Rein and Pri-Rein, respectively. The first determining time T_{D_f} and the last determining time T_{D_l} of Rein are 274.89 ms and 322.88 ms, respectively, with 47.98 ms gap time. The gap time is magnified to 56.19 ms in Pri-Rein, increasing 17.10% due to the consideration of priority in event matching.

Specifically, the average determining time is computed for each level of priority. We draw the correlation between the level of priority and the average determining time. The results are shown in Figure 11. Since subscriptions' priority is not considered in Rein, no correlation is found between the level of priority and determining time, as shown in Figure 11 (a). On the contrary, there is a strong correlation between the level of priority and determining time for Pri-Rein, as depicted in Figure 11 (b). This correlation well validates the effectiveness of Pri-Rein. Furthermore, as listed in Table 6, the determining times of Pri-Rein are much more stable than the ones of Rein for all levels of priority. On average, the standard deviations of Rein are almost 1.65 times of the ones of Pri-Rein.

Table 6: The standard deviations of determining times

Priority	Rein	Pri-Rein	Priority	Rein	Pri-Rein
0	18.10	10.59	5	18.12	10.85
1	18.13	10.63	6	18.14	11.04
2	18.11	10.85	7	18.12	11.15
3	18.12	10.84	8	18.07	11.38
4	18.18	10.77	9	18.13	11.53

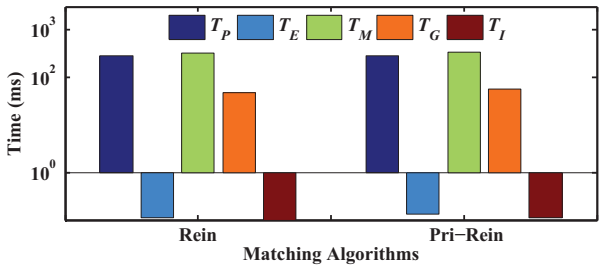


Figure 12: The preparing time (T_P), ending time (T_E), matching time (T_M), gap time (T_G) and interval time (T_I) of Rein and Pri-Rein

We conduct experiments by changing the number of subscriptions, the number of attributes, the number of constraints, the selectivity of subscriptions and the level of priority. The results are very similar to the one shown in Figure 11. Due to space limitation, these results are omitted in the paper.

6.1.2 *Efficiency*

According to the third design guideline proposed in Section 5.3, providing QoS support in event matching should not be at the price of greatly sacrificing event matching throughput. To evaluate the efficiency of Pri-Rein, we compare the preparing time, ending time, matching time, gap time and interval time of Pri-Rein with the ones of Rein. The settings of this experiment are the same as the previous one. The results are depicted in Figure 12. In fact, Pri-Rein takes a very small additional time to provide QoS support in event matching, by only 2.53% on average. The gap time of Pri-Rein is larger than that of Rein, by 17.10% on average. The reason is that the first determining time is almost same for Rein and Pri-Rein; while the last determining time of Pri-Rein is larger than the one of Rein, by about 8.15 ms. Overall, integrating the priority of subscriptions in event matching does not greatly sacrifice event matching throughput, usually less than 5.0% in most cases. The

Table 7: The standard deviations of T_P, T_E, T_M, T_G, and T_I of Rein and Pri-Rein

	T_P	T_E	T_M	T_G	T_I
Rein	11.40	0.12	12.05	3.31	0.11
Pri-Rein	10.68	0.15	11.53	2.84	0.12

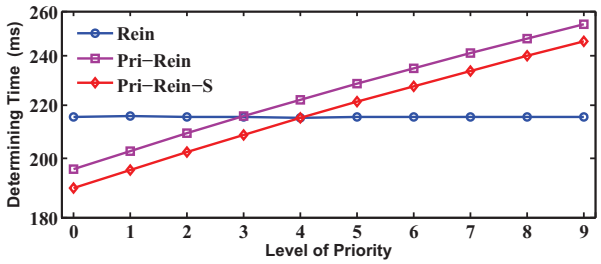


Figure 13: The determining time of Rein, Pri-Rein and Pri-Rein-S for levels of priority

standard deviations of the time metrics are listed in Table 7. There is no obvious difference between Rein and Pri-Rein.

We also conduct experiments by changing the number of subscriptions, the number of attributes, the number of constraints, the selectivity of subscriptions and the level of priority. Among these factors, the efficiency of Pri-Rein is only affected by the selectivity of subscriptions, decreasing linearly with the selectivity of subscriptions in general. For example, when selectivity is 0.1, the difference between the matching times of Rein and Pri-Rein is 2.88%; when selectivity is 0.3, the difference reaches 9.98%. Due to space limitation, these results are omitted in the paper.

6.2 Pri-Rein with Optimization

When subscriptions have different selectivity, it is beneficial to order them according to their selectivity in each level of priority, since subscriptions with high selectivity are more likely to match events and placing them in the front leads to early determination. We term Pri-Rein with this optimization as Pri-Rein-S.

6.2.1 *Effectiveness*

In this experiment, the low value and the high value of range constraints are uniformly generated from [0, 1.0], causing subscriptions with different selectivity. The average selectivity of subscriptions is 0.0825. The priority of each subscription is uniformly generated from [0, 9]. Other parameters are set as follows: $n = 500000$, $m = 50$ and $r = 5$.

The results are plotted in Figure 13. For Rein, there is no correlation between the determining time and the priority of matched subscriptions. The average determining time is almost the same for all levels of priority. For Pri-Rein and Pri-Rein-S, matched subscriptions with higher priority are picked out earlier, showing strong correlation. In Pri-Rein, subscriptions with the same priority are ordered according to the FCFS policy. Pri-Rein-S takes the selectivity of subscriptions into account when ordering the subscriptions in each level of priority. On average, the determining time of Pri-Rein-S is smaller than the one of Pri-Rein for each level of priority, by 7.18 ms about 3.20%. The standard deviations of the determining times are stable for each level of priority, about 15.70, 4.80 and 4.20 for Rein, Pri-Rein and Pri-Rein-S, respectively.

6.2.2 *Efficiency*

The settings of this experiment are the same as the previous one. We compare the preparing time, ending time, matching time, gap time and interval time of Rein, Pri-Rein and Pri-Rein-S. The results are depicted in Figure 14 with the y-axis representing time in log-scale. The standard deviations of the five time metrics are listed in Table 8

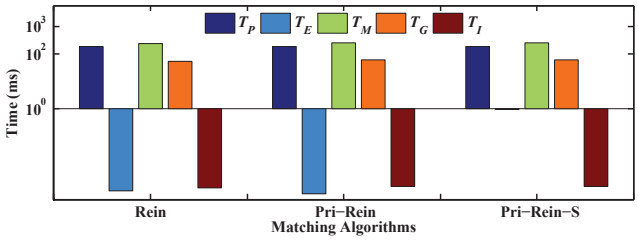


Figure 14: The preparing time (T_P), ending time (T_E), matching time (T_M), gap time (T_G), and interval time (T_I) of Rein, Pri-Rein and Pri-Rein-S

Table 8: The standard deviations of T_P, T_E, T_M, T_G and T_I of Rein, Pri-Rein and Pri-Rein-S

	T_P	T_M	T_G	T_F	T_L
Rein	5.39	5.41	0.92	5.39	5.41
Pri-Rein	4.96	4.41	0.61	4.96	4.41
Pri-Rein-S	4.75	4.20	0.85	4.75	4.11

For matching time, Pri-Rein and Pri-Rein-S take 3.64% and 3.40%, respectively, more time than Rein to match an event on average. Since Pri-Rein and Pri-Rein-S just adopt different checking order of the bitset, their preparing times are almost the same as the one of Rein. Because subscriptions with low selectivity are placed at the back in the checking order, Pri-Rein-S takes more time, about 0.98 ms, to end event matching after determining the last matched subscription, which is far larger than the ending time of Rein and Pri-Rein. Due to the consideration of subscriptions' priority, the last determining times of Pri-Rein and Pri-Rein-S are larger than the one of Rein, by 3.64% and 3.01% respectively. Accordingly, the gap times of Pri-Rein and Pri-Rein-S are larger than the one of Rein.

6.3 Discussion

In the experiments, we use range constraints to compose subscriptions. The reason to do so is twofold. First, Tama, H-tree and Rein construct their underlying data structures based on range constraints. So, it is reasonable to use range constraints to evaluate their performance. Second, since the selectivity of subscriptions affects the performance of matching algorithms, such as gap time, it is necessary to calculate the selectivity of subscriptions. Compared with other forms of constraints, it is more convenient to compute the selectivity of subscriptions composed of range constraints.

Frankly, it is challenging to design an efficient data structure for event matching that supports all data types, such as string, integer and real. A common way is to build data structures separately for numeric and string, such as in [6]. Furthermore, it is even more challenging to integrate QoS support into the underlying data structures of matching algorithms. Limited by our knowledge, it is a compromise to realize prioritized event matching based on range constraints. More efforts are needed to investigate the

problem of providing QoS support in event matching and to obtain a more general solution.

7. CONCLUSION

QoS support is critical for large-scale content-based pub/sub systems to provide guaranteed service for subscribers with high QoS requirements. To this purpose, many solutions have been proposed, much focusing on providing QoS support on routing. In addition, it is well recognized that event matching is a fundamental component in a content-based pub/sub system, attracting great attention from researchers. However, little work has been devoted to provide QoS support in event matching. In this paper, we propose the idea of prioritized event matching, aiming to integrate QoS support into event matching. By pointing out the lack of fine-grained time metrics that reveal performance detail of matching algorithms, we propose new time metrics to discover the foundation for prioritized event matching. Based on an existing matching algorithm, we make the idea proposed in the paper come true, termed as Pri-Rein. Extensive experiments have been conducted to validate the effectiveness and efficiency of Pri-Rein. Experimental results show that Pri-Rein well achieves our design goal, able to provide QoS support in event matching. More importantly, we argue that the idea proposed in this paper can be generalized to matching algorithms that are used in cloud computing or complex event processing.

8. ACKNOWLEDGMENTS

This work is partially supported by China National Science Foundation (Granted Number 61272438,61472253), Research Funds of Science and Technology Commission of Shanghai Municipality (Granted Number 14511107702, 12511502704), Research Fund of Future Network of Jiangsu Province (BY2013095-2-04). This research is also partially supported by CIP program of Morgan Stanley. Frédéric Le Mouël's work is funded by a grant from Rhone-Alpes Region, France.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC*, pages 53–61. ACM, 1999.

[2] F. Araujo and L. Rodrigues. On qos-aware publish-subscribe. In *ICDCS*, pages 511–515. IEEE, 2002.

[3] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *ICSE*, pages 443–452. IEEE, 2001.

[4] N. Carvalho, F. Araujo, and L. Rodrigues. Scalable qos-based event routing in publish-subscribe systems. In *NCA*, pages 101–108. IEEE, 2005.

[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[6] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *SIGCOMM*, pages 163–174. ACM, 2003.

[7] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *Software Engineering, IEEE Transactions on*, 27(9):827–850, 2001.

[8] M. Drosou, K. Stefanidis, and E. Pitoura. Preference-aware publish/subscribe delivery with diversity. In *DEBS*, page 6. ACM, 2009.

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[10] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD Record*, volume 30, pages 115–126. ACM, 2001.

[11] T. Fischer, A. M. Wahl, and R. Lenz. Automated quality-of-service-aware configuration of publish-subscribe systems at design-time. In *DEBS*, pages 118–129. ACM, 2014.

[12] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Proceedings of Middleware*, pages 254–273. Springer-Verlag New York, Inc., 2004.

[13] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout. Java message service. *Sun Microsystems Inc., Santa Clara, CA*, 2002.

[14] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari. Mics: an efficient content space representation model for publish/subscribe systems. In *DEBS*, page 7. ACM, 2009.

[15] K. Jayaram and P. Eugster. Split and subsume: Subscription normalization for effective content-based messaging. In *ICDCS*, pages 824–835. IEEE, 2011.

[16] Z. Jerzak and C. Fetzer. Bloom filter based routing for content-based publish/subscribe. In *DEBS*, pages 71–81. ACM, 2008.

[17] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *ICDCS*, pages 447–457. IEEE, 2005.

[18] A. Malekpour, A. Carzaniga, F. Pedone, and G. Toffetti Carughi. End-to-end reliability for best-effort content-based publish/subscribe networks. In *DEBS*, pages 207–218. ACM, 2011.

[19] P. R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *ICDCS*, pages 611–618. IEEE, 2002.

[20] S. Qian, J. Cao, Y. Zhu, and M. Li. Rein: A fast event matching approach for content-based publish/subscribe systems. In *INFOCOM*, pages 2058–2066. IEEE, 2014.

[21] S. Qian, J. Cao, Y. Zhu, M. Li, and J. Wang. H-tree: An efficient index structure for event matching in publish/subscribe systems. In *Networking*, pages 1–9. IEEE, 2013.

[22] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Networked group communication*, pages 30–43. Springer, 2001.

[23] M. Sadoghi and H.-A. Jacobsen. Analysis and optimization for boolean expression indexing. *ACM Transactions on Database Systems (TODS)*, 38(2):8, 2013.

[24] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, pages 3–5. Brisbane, Australia, 1997.

[25] M. A. Tariq, B. Koldehofe, G. G. Koch, I. Khan, and K. Rothermel. Meeting subscriber-defined qos constraints in publish/subscribe systems. *Concurrency and Computation: Practice and Experience*, 23(17):2140–2153, 2011.

[26] D. Zhang, C.-Y. Chan, and K.-L. Tan. An efficient publish/subscribe index for e-commerce databases. *Proceedings of the VLDB Endowment*, 7(8), 2014.

[27] Y. Zhao and J. Wu. Towards approximate event processing in a large-scale content-based network. In *ICDCS*, pages 790–799. IEEE, 2011.

[28] B. Zieba, M. van Sinderen, and M. Wegdam. Quality-constrained routing in publish/subscribe systems. In *MPAC*, pages 1–8. ACM, 2005.